Lightweight Techniques for Private Heavy Hitters
COM-506 Seminar Report

Tapdig Maharramli
EPFL
tapdig.maharramli@epfl.ch

Raymond Nasr
EPFL
ray.nasr@epfl.ch

12 May 2025

Abstract

This report summarizes Poplar [2], a system that solves the private heavy-hitters problem using lightweight cryptographic techniques. The system allows servers to identify popular strings across many clients without compromising individual privacy. Key contributions include malicious-secure sketching, incremental distributed point functions, and extractable DPFs. Using two non-colluding servers, Poplar achieves privacy against server misbehavior while requiring only a single message from each client and no public-key cryptography. Experimentally, with 400,000 clients holding 256-bit strings, Poplar identifies the 200 most popular strings in 54 minutes with high parallelizability, providing an efficient solution for privacy-preserving data collection.

1 Introduction

Organizations seeking to improve their products often need to collect user data, but traditional methods risk exposing sensitive information [2]. Consider browsers collecting crash-causing URLs, electric car companies tracking battery drain locations, or app developers measuring engagement. Poplar addresses these scenarios through the private heavy-hitters problem: identifying strings exceeding a popularity threshold without exposing individual data.

Unlike approaches based on general-purpose multiparty computation [8], secure aggregation [5], or local differential privacy [1], Poplar uses two non-colluding servers to provide strong privacy with concrete efficiency. The system protects privacy against arbitrary misbehavior by one server while ensuring correctness against malicious clients. Each client sends only a single message.

Poplar introduces several lightweight cryptographic techniques: incremental distributed point functions, malicious-secure sketching, and extractable DPFs. These techniques not only improve efficiency but may benefit other privacy-preserving applications. While Poplar reveals slightly more information than just the heavy hitters, this leakage is precisely quantified and can be mitigated through differential privacy [6].

2 Problem Statement

Poplar operates in a setting with two data-collection servers and C participating clients. Each client i, for $i \in \{1, \ldots, C\}$, holds a private input string $\alpha_i \in \{0,1\}^n$. The system aims to compute useful aggregate statistics over these private strings while minimizing information leakage about individual strings [2].

Two primary tasks are addressed:

1. **Subset Histogram**: The servers hold a set $S \subseteq \{0,1\}^n$ of strings and want to learn, for each string $\sigma \in S$, how many clients hold that string. This set may be public or kept secret from clients.

2. **Heavy Hitters**: For a threshold $t \in \mathbb{N}$ such that $1 \leq t \leq C$, where C is the number of clients, the servers want to identify all strings that appear in the client dataset at least t times (t-heavy hitters).

The communication pattern in Poplar consists of three phases:

- **Setup**: Servers generate and distribute public parameters to clients.
- **Upload**: Each client sends a single message to each server (Server 0 and Server 1).
- **Aggregate**: The two servers run a protocol between themselves to compute the desired aggregate statistics.

Poplar provides several key security properties:

- **Completeness**: With honest participants, servers correctly learn the aggregate statistics.
- **Robustness to malicious clients**: Malicious clients cannot bias results beyond choosing their own input arbitrarily.
- **Privacy against a malicious server**: If one server is honest, a malicious server learns nothing about client data beyond the aggregate statistics.

For the heavy-hitters protocol, Poplar reveals slightly more information, characterized by a leakage function L. This leakage depends only on the multiset of strings held by honest clients, without revealing associations between clients and strings, and scales logarithmically with the number of clients when the threshold is a fixed fraction.

3 Background

This section reviews existing techniques for private aggregation that serve as a foundation for Poplar. The *Private Subset Histogram problem* requires servers holding a set $S = \{\sigma_1, \ldots, \sigma_m\} \subseteq \{0,1\}^n$ to learn, for each $\sigma \in S$, the number of clients holding σ, without revealing S or learning about strings outside S. Distributed Point Functions (DPFs) are a cryptographic tool used here; a DPF allows secret-sharing a high-dimensional vector with a single non-zero entry at point $\alpha \in \{0,1\}^n$ with value $\beta \in \mathbb{F}$, using compact $O(n)$-size keys.

A simple protocol for Private Subset Histograms utilizes DPFs. Clients generate keys $(k_{i0}, k_{i1}) \leftarrow Gen(\alpha_i, 1)$ for their string α_i and send one key to each server. Servers, for each $\sigma_j \in S$, compute the sum of their respective keys evaluated at σ_j, $val_{jb} = \sum_{i=1}^{C} Eval(k_{ib}, \sigma_j)$. The total count for σ_j is $val_{j0} + val_{j1}$, which equals $\sum_{i=1}^{C} 1\{\alpha_i = \sigma_j\}$ due to DPF correctness.

4 Malicious-Secure Sketching

This section details methods to achieve robustness against malicious clients and servers in private subset histogram computation. Prior work attempted to harden the simple DPF scheme by checking if client-submitted keys were well-formed, verifying that they expand to shares of a vector that is 0 everywhere and 1 at a single position within the queried subset S. Servers would compute $\overline{v}_b = (Eval(k_b, \sigma_1), \ldots, Eval(k_b, \sigma_{|S|}))$ and check if $\overline{v}_0 + \overline{v}_1$ was a valid weight-one vector over S.

Existing sketching protocols, however, had limitations. They often lacked protection against malicious servers, or required additional client interaction or servers. More subtly, the check on the subset S alone was insufficient against a malicious client who could craft keys that evaluated to 1 on multiple strings outside S but correctly on only one within S, thereby gaining undue influence. This vulnerability highlighted the need to ensure that keys represent a sharing of a vector with at most one "1" at any relevant position in the entire universe, addressed by techniques like Extractable DPFs.

A *new lightweight malicious-secure sketching protocol* is introduced to handle malicious servers. The client encodes their vector $\overline{v}$ as $(\overline{v}, \kappa\overline{v})$ using a random κ and provides additive shares of this pair to the servers, along with correlated randomness for a secure computation. Servers compute sketches $(z_b = \langle \overline{r}, \overline{v}_b \rangle,\ z_b^* = \langle \overline{r}^*, \overline{v}_b \rangle,\ z_b^{**} = \langle \overline{r}, \overline{v}_b^* \rangle)$ using jointly sampled random vectors $\overline{r}, \overline{r}^*$. They then perform a two-round secure computation to check if $(z_0 + z_1)^2 - (z_0^* + z_1^*) + \kappa(z_0 + z_1) - (z_0^{**} + z_1^{**}) = 0$. Client-provided shares of derived coefficients enable this check on public values. This protocol provides robustness against malicious servers tampering with the encoded vector, resulting in detection with high probability.

Complementing this, *Extractable DPFs* are introduced to address the weak protection against malicious clients. Based on a refined analysis of the DPF construction in the random-oracle model, Extractable DPFs make it computationally infeasible for a malicious client to generate keys with identical public parts that evaluate to 1 at more than one position known to the client. This allows servers to infer that well-formed keys represent shares of a vector with at most one "1" at a relevant index across the entire domain, preventing malicious clients from influencing multiple counts.

5 Incremental Distributed Point Functions

Incremental Distributed Point Functions (IDPFs) are a key contribution of Poplar that significantly improves efficiency for the heavy-hitters protocol. Standard Distributed Point Functions (DPFs) [7, 3] provide a way to secret-share a vector of dimension 2^n that is non-zero at only a single point. IDPFs extend this idea to efficiently share values on nodes of a binary tree with 2^n leaves, where only a single path has non-zero values [2].

Formally, an IDPF scheme consists of:

- **Gen**$(\alpha, \beta_1, ..., \beta_n) \rightarrow (k_0, k_1)$: Given a string $\alpha \in \{0,1\}^n$ and values $\beta_1 \in G_1, ..., \beta_n \in G_n$, outputs two keys representing secret shares of a tree with a single non-zero path ending at leaf α.

- **Eval**$(k, x) \rightarrow \cup_{i=1}^{n} G_i$: Given an IDPF key k and string $x \in \cup_{i=1}^{n} \{0,1\}^i$, outputs a secret-shared value for the node identified by x.

The correctness property ensures that when combining outputs from both keys, the result is β_i if x is a prefix of α with $|x| = i$, and 0 otherwise. The security property ensures that an adversary seeing only one key learns nothing about α or the values $\beta_1, ..., \beta_n$.

While standard DPFs could implement this functionality with $O(n^2)$ key size and evaluation time, the paper's direct IDPF construction achieves $O(n)$ complexity. This optimization is crucial for the heavy-hitters protocol's efficiency, as it reduces client-to-server communication from quadratic to linear in the string length n.

The IDPF construction extends the DPF approach from [3] with additional steps to support prefix outputs efficiently. For each level ℓ, the construction incorporates an extra intermediate step to generate an output share for that level while maintaining security. The implementation optimizes further using AES hardware instructions to implement the underlying PRG operations.

6 Poplar Heavy-Hitters Protocol

The Poplar heavy-hitters protocol efficiently identifies t-heavy hitters (strings held by more than t clients) without compromising individual privacy. The approach uses incremental DPFs and proceeds in two key steps:

First, the protocol implements private prefix-count queries, which reveal how many client strings begin with a given prefix without exposing individual strings. Using these queries, the servers can identify heavy hitters through a breadth-first search of the prefix tree, pruning branches that cannot contain heavy hitters [4, 1].

Each client i uses incremental DPFs to create secret shares of a tree that is zero everywhere except for nodes along the path to the client's string α_i. When the servers want to count strings with prefix p, each server evaluates all client DPF keys at p and publishes the sum. Adding these sums reveals exactly how many client strings start with prefix p.

The protocol traverses the prefix tree level-by-level. At each level ℓ, the servers identify prefixes that appear in at least t strings. They then extend these prefixes for the next level and continue. After n iterations (for n-bit strings), the servers identify all t-heavy hitters.

To protect against malicious clients, Poplar uses the malicious-secure sketching and extractable DPFs. At each level of the tree, servers verify that client-provided values represent valid shares before incorporating them into the aggregate [2].

For longer strings ($n \gg \lambda$), Poplar offers a hashing-based optimization that reduces client communication from $O(\lambda n)$ to $O(\lambda^2 + n)$ bits and improves round complexity from $O(n)$ to $O(\lambda)$.

The protocol's security analysis proves that even with a malicious server, an adversary learns only information characterized by the leakage function L, which depends only on the multiset of honest client strings without revealing client-string associations.

7 Leakage Mitigation and Differential Privacy

The heavy hitters output itself or intermediate prefix counts can leak sensitive information. To bound adversary inference, Poplar can optionally provide ϵ-differential privacy.

This is achieved by adding Laplace noise to the outputs of the prefix-count oracle queries. In Poplar protocol, when server b computes $val_{p,b}$ for prefix p, it samples noise $v_{p,b} \leftarrow Laplace(1/\epsilon)$ and publishes $val'_{p,b} = val_{p,b} + Round(v_{p,b})$. The noised sum $val'_{p,0} + val'_{p,1}$ is used.

Using advanced composition ($q = nC/t$ queries), ϵ-DP per query yields (ϵ', δ')-DP overall. Noise adds error; its magnitude $2\lambda/\epsilon$ is $\leq e^{-\lambda}$ per query. If $2\lambda/\epsilon < 0.05t$, noise is unlikely to cause correctness failures. Poplar's noise ($O(1/\epsilon)$ error) is lower than local DP ($\Omega(\sqrt{C}/\epsilon)$). An example setting with $n = 256, \delta' = 2^{-40}, t = 0.01C$ and per-query $\epsilon = 0.001$ gives overall $(1.22, 2^{-40})$-DP with manageable noise relative to the threshold for large C.

8 Implementation and Evaluation

This section details the practical implementation and experimental evaluation of the Poplar system. Poplar is implemented in $\approx 3,500$ lines of Rust code, publicly available. Sketching uses specific field sizes for security guarantees.

Experiments were conducted on Amazon EC2 c4.8xlarge instances in N. California and N. Virginia (61.8ms RTT), using 400,000 clients with 256-bit strings from a Zipf distribution, finding heavy hitters $> 0.1\%$.

Client costs show linear scaling for key generation time and size ($O(\lambda n)$), outperforming baselines. Server communication per client is tens of kilobytes, scaling linearly with n, and is orders of magnitude cheaper in dollar cost than a standard DPF baseline. End-to-end time for 400k clients is $\approx$ 53 minutes, scaling linearly with client count. The system is highly parallelizable, capable of processing millions of clients efficiently with sufficient machines per server. The evaluation robustly supports the claims of lightweightness and practical efficiency.

9 Conclusion

Poplar demonstrates that efficient privacy-preserving data collection is achievable with lightweight cryptographic techniques. By combining novel cryptographic primitives with careful protocol design, the system creates a balance between strong privacy guarantees and practical performance. Unlike approaches requiring heavyweight cryptography or extensive client-server interaction, Poplar achieves its goals with minimal communication overhead. The significant performance improvements over previous methods make privacy-preserving analytics feasible at scale, opening new possibilities for organizations to gather valuable insights while respecting user privacy.

References

[1] Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Guha Thakurta. Practical locally private heavy hitters. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[2] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776. IEEE, 2021.

[3] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303. ACM, 2016.

[4] Graham Cormode, Flip Korn, Shanmugavelayutham Muthukrishnan, and Divesh Srivastava. Finding hierarchical heavy hitters in data streams. In *Proceedings of the 29th international conference on Very large data bases*, pages 464–475. VLDB Endowment, 2003.

[5] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 259–282, 2017.

[6] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, pages 265–284. Springer, 2006.

[7] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. *EUROCRYPT*, pages 640–658, 2014.

[8] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, 1987.